Interactive Fractal Compositions

Felix Haase
Welfenlab
Leibniz University Hannover
[*]

Maximilian Klein
Welfenlab
Leibniz University Hannover
[†]

Andreas Tarnowsky
Welfenlab
Leibniz University Hannover
[‡]

Franz-Erich Wolter
Welfenlab
Leibniz University Hannover
[§]

Abstract

With the ever rising quality and complexity standards in computer graphics, the generation of detailed content has become a bottleneck. While high quality visualization can be achieved at comparatively low cost, content generation remains a labor intensive and expensive task. Procedural approaches can support this process by automating parts of it.

One common problem of procedural methods is that the variables controlling the result are difficult to adjust. Especially fractals may have unintuitive parameters, which make them difficult to handle in praxis.

In this paper we introduce the concept of a *Region Tree* to structure the workflow with these procedures and present a supporting framework. On top of this we show how we used GPU integration to make interactive editing possible.

With this approach it is easy to construct any number of procedural models from a set of user defined characteristics. We illustrate our method by creating an earth-like complex planet completely procedurally.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Fractals and Virtual reality I.3.5 [Computer Graphics]: Methodology and Techniques—Graphics data structures and data types and Interaction techniques;

Keywords: Multi-resolution and Multi-scale Methods, Real-time Rendering, Rendering and Visualization of Large-scale Models, Procedural Methods

[*]fhaase@welfenlab.de
[†]mklein@welfenlab.de
[‡]atarnows@welfenlab.de
[§]few@welfenlab.de

1 Introduction

The generation of high detailed objects became increasingly important during the last years. The visual appearance is particularly important in areas where the impact of the product depends on the quality of the visualization, as in the advertising, film or game industry. However creating those details manually can be a time consuming task and the possible level of detail is limited. Fully automated processes on the other hand are not feasible because of the creative nature of design processes in general. A common solution is to use semi-automated approaches in which the user guides the automated process. This is especially useful when the time needed for the automated process allows the user to control it interactively.

One way to generate details automatically is to use fractals, as they are not bound to any level of detail. Noise based fractals are particularly useful for a seed based generation of details. Those fractals can be used to generate randomized but controllable details by adjusting the seed value. We will concentrate on the generation of landscapes in this paper, but our methods are not restricted to them.

1.1 Related Work

Noise functions are a common approach to generate procedural textures and models for graphical applications. They are often used to recreate natural phenomena where a certain amount of irregularity is needed to achieve a realistic look. The original Perlin noise function was proposed by Ken Perlin in 1985 [Perlin 1985]. Because of its computational complexity, it has been subject to several optimizations since. For example, Perlin proposed Simplex Noise [Perlin 2001] as an improvement to his former algorithm. This algorithm has several advantages over the original, e.g. offering a reduced complexity in higher dimensions. There are also GPU implementations of the noise function [Tzeng and Wei 2008; Olano 2005], one particular fast implementation has been presented in GPU Gems 2 [Green 2005]. It makes use of massive parallelization and the interpolation abilities of modern graphic cards.

Fractals can be created by combining noise functions at different scales. An overview about this topic can be found in [Mandelbrot 1987] and [Miller 1986].

For the procedural generation of terrain many different approaches exists.

One way is to simulate different types of erosion such as wind [Shao 2008] or water flow [Kelley et al. 1988; Musgrave et al. 1989]. Although this approach is most promising in terms of realistic results, the complexity of the underlying natural phenomena makes a convincing implementation very difficult and time consuming. Current implementations require a lot of computing time and thus cannot be used in interactive tools.

Another way is to use genetic programming as in [Frade et al.

2009]. The idea is to compose a set of mathematical functions to create different shapes and then let the user pick the most appropriate one. Based on this selection the genetic program is trained and new shapes are generated. This process is repeated until the resulting shape is good enough. Unfortunately this procedure can take a very long time and the resulting landscapes tend to look artificial.

Another procedural method is to use fractals like fBm (*fractional Brownian motion*) or the multifractals presented in [Ebert et al. 2003]. Usually a composition of different fractals is needed to achieve the variety of natural terrains. This leads to a significant amount of variables with often unintuitive influences on the look of the landscape. Different commercial tools exist to create and render planets with this approach. The Terragen software [Planetside-Software 2012] renders visually impressive results, but it is not an interactive application and the included preview function is very limited. Because the result can only be validated after minutes of rendering, the adjustment of fractal parameters is a time consuming procedure. In 2006, Schneider et al. [Schneider et al. 2006] presented an interactive fractal editor. The user can edit the different parameters of a single fractal, such as lacunarity or the number of octaves, and add custom deformations. Another GPU-driven realtime terrain generator is Litosphere [Bösch 2012], they use a graph to allow the user to edit the terrain in realtime.

Hnaidi et al. [Hnaidi et al. 2010] presented an algorithm that can create rich terrains based on user defined feature lines using the diffusion equation. They present visual appealing results, but their method requires the user to create all the feature lines manually which results in a time consuming process.

To render a procedurally generated object, one has to apply textures or colors and calculate shadows and light onto the object. Dachsbacher presented in [Dachsbacher 2006] a layer-based approach to colorize a terrain represented as a heightfield.

1.2 Our Contribution

We contribute a new way to create highly detailed objects with a technique that requires a description of details and a seed value to generate enriched versions of the model. Our *Fractal Compositing* technique combines the intuitiveness of a simple base mesh with the possibility to create high details with unlimited diversity. In order to demonstrate our method, we present a fractal planet generator that is capable of but not limited to visualizing earth-like planetary surfaces with different climate zones and terrain types.

Within this context, we also present our *recursive Brownian distortion* (rBd) algorithm, a modification of the *fractional Brownian motion* (fBm) fractal generation process. It allows the creation of more realistic and diversified coastlines and thus more realistic earth-like planets.

All our techniques are able to provide interactive framerates when editing planets, even with high amounts of detail. This is especially useful for design processes where unintuitive parameters have to be adjusted, since the results are immediately visible.

To this end, we present how we extended the GLSL shading language to accommodate for this interactivity in an easily accessible way, without loosing any performance.

In the remainder of this paper we will present our *Fractal Compositing* idea together with a semi-automated tool that allows the user to interactively create a description of the models details. We demonstrate our method based on a planet generator that we used to generate earth-like planets and compare our results to the NASA Visible-Earth high quality images [NASA 2004].

2 Fractal Compositing

We introduce a technique that we call *Fractal Compositing*. This technique uses fractals to generate our enriched mesh based on a simple mesh. In this method we utilize the fractals in two different ways to generate the enriched mesh.

1. As a heightfield which deforms the object.

2. As a selector for regions or properties.

Using fractals as heightfields is a method that is widely used, resulting in astonishing results. Previous approaches tended to create very specialized heightfields which were primarily fit for a single purpose, but not for a mesh that requires a broader diversity of features.

Instead, we use different fractals to generate regions and to assign properties to our mesh. Those regions can be enriched by a heighfield or they can be refined in further regions etc.. The major advantage of our method is the ability to use fractals which are good at generating a certain detail locally, without being bound to the fractal globally.

We will demonstrate our idea in the next sections with our implementation of a fractal planet generator based on the *Fractal Compositing* idea with a few extensions for creating further details procedurally or using user generated details.

3 Planet Generation

In order to demonstrate the flexibility and ease of use of our concept, we present an application for modeling and rendering realistic planetary surfaces in realtime using OpenGL.

To create such a planet, we start with a spherical mesh and displaced the vertices using a combination of different shader functions. The vertex- as well as the fragment-shader use a weighted combination of different fractals and the contributions of those to the final result are represented by a tree structure that we call a *Region Tree*, which will be explained in the next section. In order to increase the rendering quality the displacement was done in a per-pixel fashion. Thereby the color calculation and the overall shading of the surface adapts well to the situation.

In addition to base-fractals that define the general distribution of features of the planet, a variety of different fractal types is used for modeling the individual regions on the planet – such as forests, deserts and mountain regions. Many of these fractal types are commonly known as e.g. *fractional Brownian motion* (fBm), *Multifractal* (MF) and *Ridged Multifractal* (RMF). All these fractals are based on certain *noise functions*. A common approach that yields good results is to use the aforementioned *Perlin noise*. However, in our example we use *Simplex noise* that needs less computational effort while providing a *better* visual quality since it offers a higher isotropy [Perlin 2001].

For creating the initial planetary surface, fractional Brownian motion is a very popular approach. However, as Schneider et al. already pointed out, surfaces created this way look relatively homogeneous when applied to larger scales [Schneider et al. 2006] – which is obviously the case when modeling a whole planet.

Therefore we used a modified version of the original fBm algorithm which produced more variety and is explained in the next section.

3.1 Coastlines and continents

In order to compensate for the homogeneity of the fBm method, we use another fractal algorithm that is based on a technique known as *Domain warping* [Schneider et al. 2006]. In this work, we call our specific version of this algorithm *recursive Brownian distortion* (rBd). The base function for this kind of fractal still consists of *fractional Brownian motion*. However, the input vector (that is a 3D point of the planetary surface lying on the unit sphere) is warped in a specific way before the fBm algorithm is applied. The warping is applied as follows:

```
float rBd(vector3 position, int depth) {
    const float weight = 0.2;
    int seed = 0;
    float falloff =
        fBm(position * variation, seed);
    for(i = 0; i < depth; i++) {
        vector3 distortion = vector3(
            fBm(position, seed + c1),
            fBm(position, seed - c2)),
            fBm(position, seed));
        position += weight * distortion;
        weight = weight * falloff;
        seed += 500; // Some random value
    }

    return fBm(position, seed);
}
```

With `fBm(vector3, float offset)` being the *fractional Brownian motion* method using a fixed value for lacunarity, persistence and the number of octaves. The values for `weight` and `variation` have been chosen empirically in order to reflect the structure of the earth's coastlines as closely as possible. The constants `c1` and `c2` shift the seed of the fBm method to generate a better warping effect.

As shown in *figure 1*, the coastline created with this method looks more natural than the simple fBm approach. Although the artificial planetary surface generated with our rBd method lacks the local details of a real planet, its coastline appears to be quite convincing.

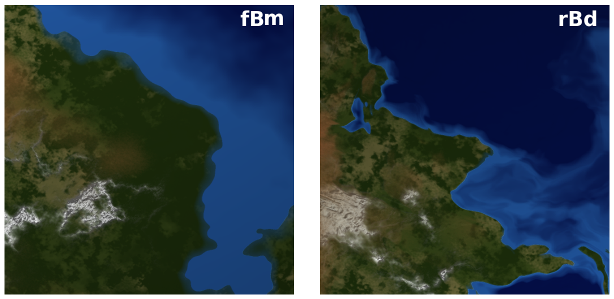

Figure 1: *Comparison of the fBm and our rBd method. As both methods create completely different planets we chose two typical coasts. The algorithms differ only in the use of the fBm and rBd method. The right image replaces the use of the fBm method with the rBd method to calculate the coastline and inner continental features.*

3.2 Region Tree

The *Region Tree* is a data structure that contains information about the distribution of distinct surface characteristics of a model, e.g different terrain types for a planet generator.

Figure 2 shows an exemplary version of such a tree used for rendering the images within this paper.

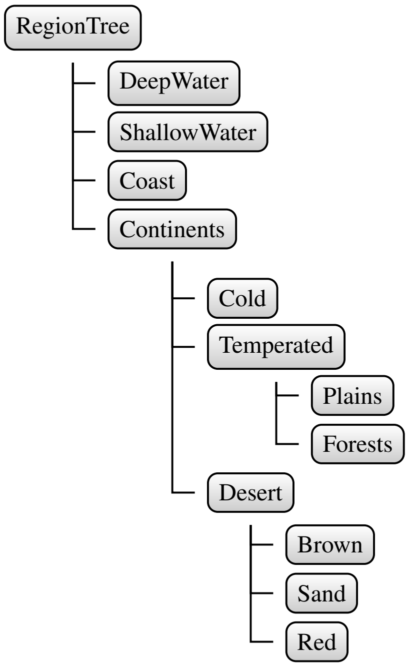

Figure 2: *The* Region Tree *used for rendering the example images. The weights for the different regions differ in the examples and are not shown in this illustration.*

The different surface characteristics are hierarchically organized by the tree structure. By using this strucure the user can seperate different large scale areas before concentrating on the details of every one of them individually.

Each node represents an area of the surface and the children break down this area into parts with the size indicated by their weighting factor. This means that the inner nodes of the tree define the distribution of the underlying nodes by a weighting factor and only the leaf nodes determine the actual manifestation of the surface.

This manifestation is applied by functions displacing the height-field of the respective feature region. Each of these methods is a (shader-) function $f : \mathbb{R}^3 \to \mathbb{R}$ that may contain a combination of completely user defined fractals. In order to calculate the final color of the surface, the user can use the weights in addition to height and slope information.

To prevent sharp edges between the children, the user can influence the smoothness of the transitions. This information is stored in a decision texture which is used to quickly decide which area a surface point lies in. Each point can be located in several areas which are then interpolated according to the percentage of each, a feature providing smooth transitions.

As a consequence of the tree structure each region can only be adjacent to its left or right sibling. If there is no left or right sibling, it will be adjacent to one of the parents siblings. This constraint can be used to fulfil certain logical neighbouring constraints, such as having coasts lie between oceans and continents or prevent forests from being adjacent to deserts.

It would be possible to use a more general graph structure instead of a tree to store all this information. One example for such an approach is used in [Bösch 2012]. But the restrictions of a tree makes the design of the surface more managable because the neighbouring conditions are easily visible and the strict hierarchy is maintained automatically.

A simple method we used to generate earth-like planets was to generate a distorted solar energy map and combine it with a fractal rain map. This gives us a value from which we can generate climes as shown in figure 3.

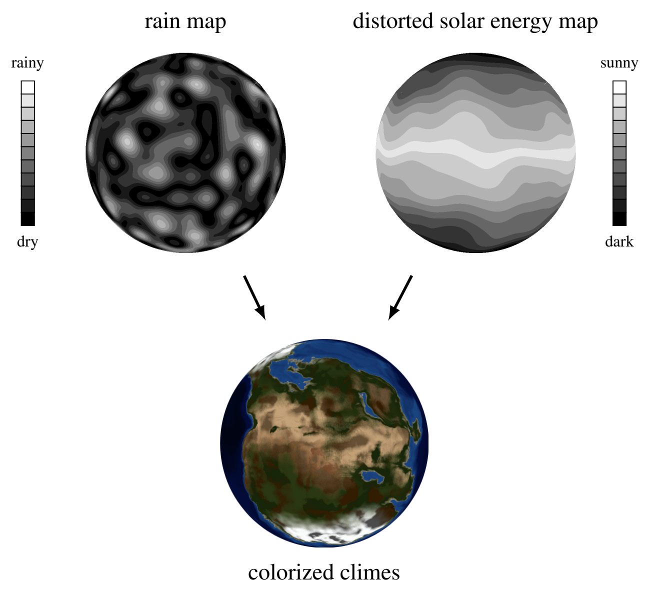

Figure 3: *Method for generating climes using a distorted solar energy map and a fractal rain map*

Although this method does not use physical information about the elevation and wind direction, we find that it produces realistic looking results.

3.3 Layer Extension

In some cases it can be necessary to make specific adjustments to the resulting model, e.g. to add man-made structures. We allow adding textures to the surface of the mesh, which can be used at will, for example as weights or as an additional elevation map.

Furthermore, one can access all the generated textures and run supplementary programs on this data. This makes it possible for a user to access their own information to adjust the terrain at their will. It also allows a programmer to extend the framework with features that need the generated textures of the preliminary result, e.g. creating river beds. This method is capable of supporting additional applications which are impractical to implement using only the *Region Tree*.

One example of such a layer extension is shown at the end of the paper in *figure 7*.

The *Layer Extension* and the *Region Tree* are particular useful when generating planets, but these techniques are not limited to it. The *Region Tree* approach represents a generic structure that can describe features and deformations for arbitrary objects via fractals and mathematical functions. Our *Layer Extension* is a simple way to support discrete input data for the *Region Tree* like textures.

4 Compositing Framework

One of the most important parts of our work is the compositing framework. Working with fractals to create realistic planets requires lots of experimenting to find the right parameters. Our goal was to create a framework that allows to simply define new parameters and change them interactively. To achieve this, we decided to make extensive use of shaders to get visually pleasing results and a powerful tool.

We extended the GLSL shading language by creating a preprocessor that allowed us to include files. Furthermore, we introduced the keyword `#param` that allows the user to define a parameter that can be controlled in the framework. For each of these parameters a uniform variable is created in the shader that is updated automatically by our application. The whole GLSL compilation unit is shown in *figure 4*.

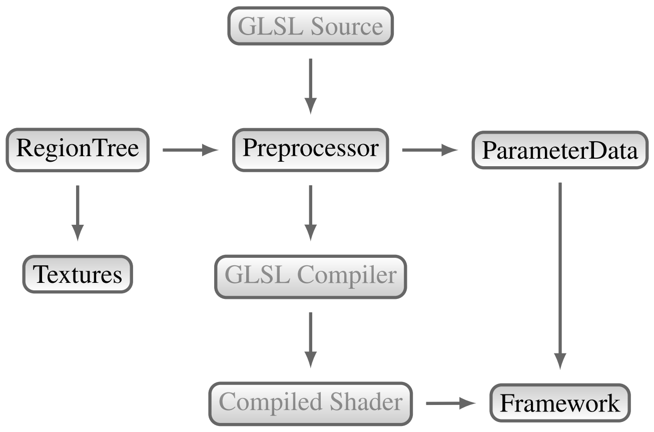

Figure 4: *Our customized GLSL compilation Unit with a preprocessor and the* Region Tree *from which we generate code and interpolation textures*

The *Region Tree* mentioned in *section 3.2* is used to generate code that calculates the different weights. The *Region Tree* itself is not powerful enough to generate all the code needed as mentioned in *section 3.2*.

The tree structure is especially useful in terms of usability and performance. The user can build a simple description for the object he wants to create and then our *Region Tree* generates a decision texture for the description, lowering the number of branches, like `if` instructions, in the shader code. As a lot of branches are known to lower performance our texture based approach proved to be a fast and reliable tool for us.

5 Results

All the examples we present in this paper were rendered on a workstation with an NVIDIA GeForce GTX 480 graphics processor. The fractals and the coloring were computed on the graphic card using GLSL. We used no textures, all colorings are based on the *Region Tree* and the underlying fractals.

Our Region Tree approach allowed for a fast adaption of planetary features. *Figure 6* shows how we adapt the overall temperature of the planet by adjusting three weights in the *Region Tree* shown in *figure 2*. We adapted the percentages of the *Cold*, *Temperated* and the *Desert* areas.

To emphasize the realism of our coastlines we compared our results to an image of the earth. We took the NASA visible earth

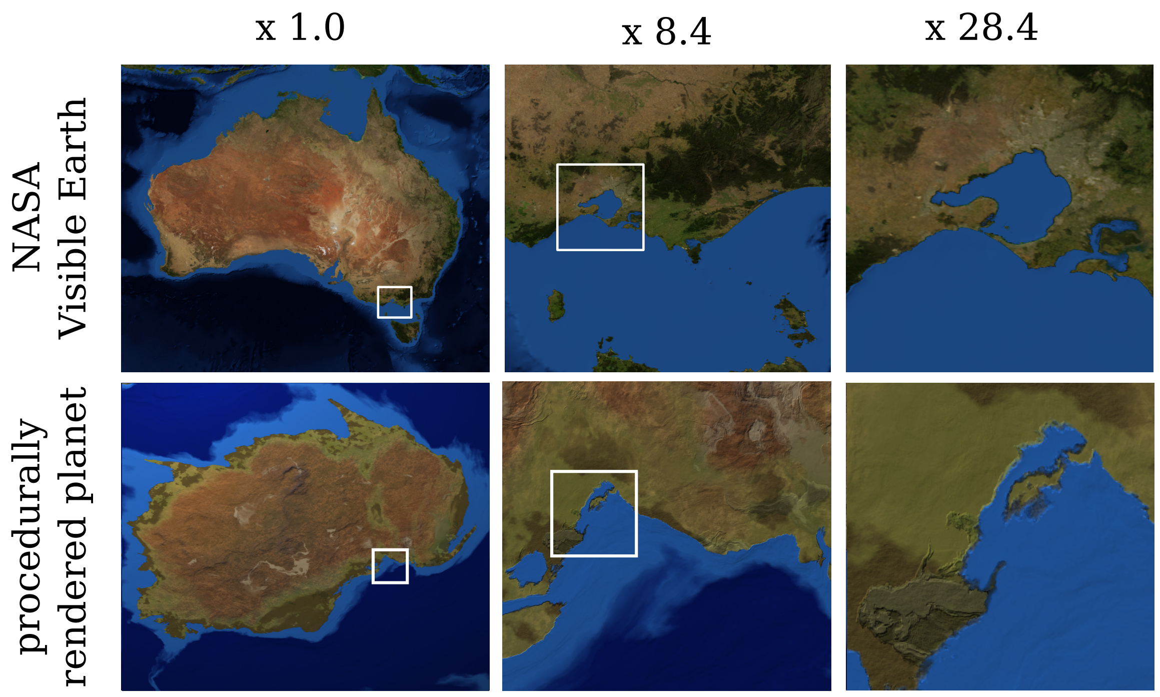

Figure 5: *Comparison of an actual image of the earth with our fractal algorithm. Both use only one high resolution image (*$\sim 11000 \times 9000$ *pixels). We created a procedurally generated planet that has a continent like Australia and magnified a region in both images using the same magnification factor.*

image [NASA 2004] of Australia and magnified the bay near Melbourne. We compared different magnifications of this real world image to one we created using our framework. *Figure 5* shows both versions and even though the level of detail of the NASA images is higher with an magnification factor of 28, our method is capable of creating earth-like coastlines.

The resulting framerates are highly dependent on the number of octaves used in the fractal calculation. Thus we defined the parameter *Fractal Detail* to control the number of octaves. To increase the performance, some features have a diminished level of detail compared to others, e.g. the number of octaves for the coastlines is always equal to the *Fractal Detail*, whereas the forest contours have only half the octaves.

In *table 1* the number of noise function calls per region is listed. The numbers are based on a Fractal Detail level of 6. The fragment-shader has to call the different fractal codes for each visible pixel on the planets surface.

The minimal number of noise function calls for a pixel is 150 for an ocean region, whereas the maximal number is 414, the sum of all calls. The blending of climes can force the shader to calculate all fractal types at once, but it is very unlikely to calculate all of them for one pixel.

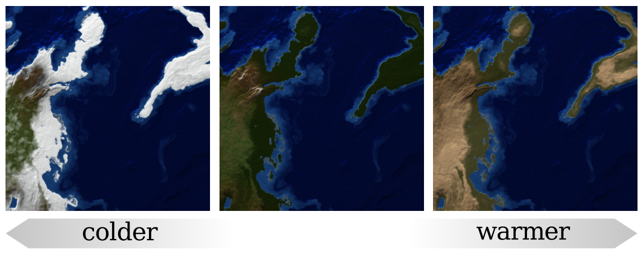

Figure 6: *The climatic zone of the planet can be changed easily by adjusting the tree weights. The effect is immediately visible.*

Table 1: *Number of* `noise` *function calls per region for a Fractal Detail of 6. The base fractal is used in the first node of the* Region Tree *cf.* figure 2. *Regions that are not shown, like the ocean region, do not call the* `noise` *function*

Regions	Maximal Calls
BaseFractal	150
Continents	18
Temperated	6
Plains	21
Forests	3
Desert	162
Brown	21
Sand	3
Red	30

The average number of calls depends highly on the chosen *Region Tree*, the seed value and the view angle. We created an equatorial orbit around the earth and measured the average framerate for the planet shown in *figure 7*. The diagram in *figure 8* shows the framerate for different Fractal Detail levels. We achieve interactive framerates up to a Fractal Detail level of 6, which produces visually appealing results.

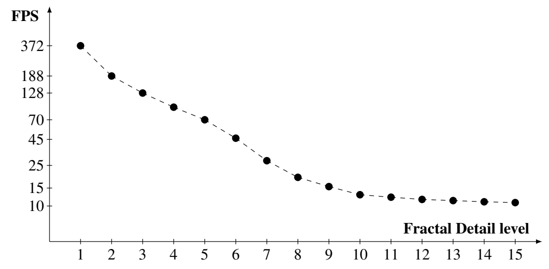

Figure 8: *Frames per second (FPS, shown in logarithmic scale) for rendering the full fractal planet with different numbers of octaves on* 800×600 *pixels.*

6 Conclusion and Future work

We have presented a new approach to support the construction and modification of procedural models. By introducing the Region Tree, we allow the developer to structure complex procedural descriptions in consecutive steps.

Our supporting framework utilizes the processing speed of modern GPUs to achieve interactive framerates. This reduces the time for the adjustment of the procedural parameters, because the results are immediately visible.

The proposed system is not limited to the creation of planetary surfaces. It can be altered to enrich more complex base meshes with completely different details like scars or birth marks on humans or other irregularities on natural surfaces.

Another way to improve our work is an intensive use of our layer extension. More complex procedural methods could be implemented to enrich the created meshes e.g. by rivers or man made structures.

References

BÖSCH, F. 2012. Lithosphere homepage. *Online at* `http://lithosphere.codeflow.org/`*, accessed 13-August-2012.*

DACHSBACHER, C. 2006. Interactive terrain rendering: Towards realism with procedural models and graphics hardware. *Perspective.*

EBERT, D., WORLEY, S., MUSGRAVE, F., PEACHEY, D., AND PERLIN, K. 2003. Texturing and modeling, 3rd edition. *Morgan Kaufmann Publishers.*

FRADE, M., FERNANDÉZ DE VEGA, F., AND COTTA, C. 2009. Breeding terrains with genetic terrain programming: the evolution of terrain generators. *International Journal of Computer Games Technology.*

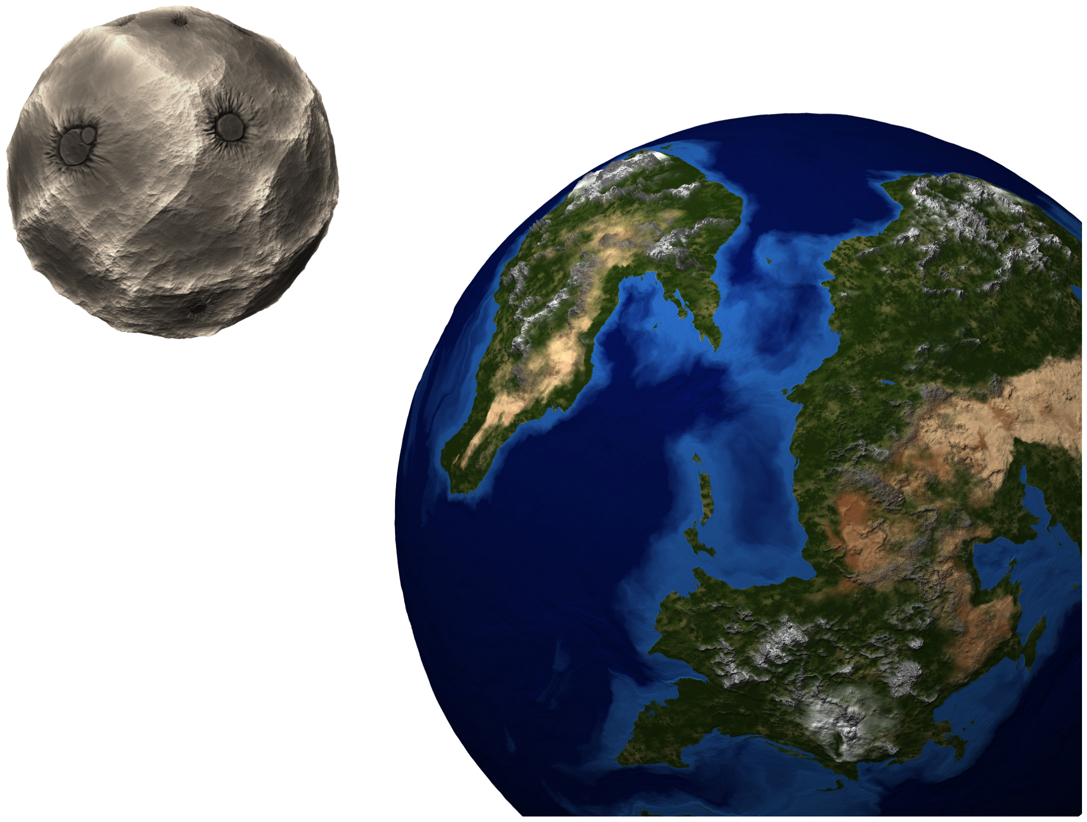

Figure 7: *High resolution rendering of an earth-like planet with a moon. The displacements are exaggerated to improve the plasticity. The craters on the moon are user generated. We added them using our layer extension.*

GREEN, S. 2005. Implementing improved perlin noise. *GPU Gems 2*, 409–416.

HNAIDI, H., GURIN, E., AKKOUCHE, S., PEYTAVIE, A., AND GALIN, E. 2010. Feature based terrain generation using diffusion equation. *Computer Graphics Forum (Proceedings of Pacific Graphics) 29*, 7, 2179–2186.

KELLEY, A., MALIN, M., AND NIELSON, G. 1988. *Terrain simulation using a model of stream erosion*, vol. 22. ACM.

MANDELBROT, B. 1987. *Die fraktale Geometrie der Natur*. Birkhäuser Basel.

MILLER, G. 1986. The definition and rendering of terrain maps. *ACM SIGGRAPH 86 Computer Graphics 20*, 4, 39–48.

MUSGRAVE, F., KOLB, C., AND MACE, R. 1989. The synthesis and rendering of eroded fractal terrains. In *ACM SIGGRAPH 89 Computer Graphics*, vol. 23, ACM, 41–50.

NASA. 2004. Nasa visible earth. *Online at* `http://visibleearth.nasa.gov/view.php?id=73884`*, accessed 08-August-2012*.

OLANO, M. 2005. Modified noise for evaluation on graphics hardware. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, ACM, 105–110.

PERLIN, K. 1985. An image synthesizer. In *ACM SIGGRAPH 85 Computer Graphics*, vol. 19, ACM, 287–296.

PERLIN, K. 2001. Noise hardware. *Real-Time Shading SIGGRAPH Course Notes*.

PLANETSIDE-SOFTWARE. 2012. Terragen homepage. *Online at* `http://www.planetside.co.uk`*, accessed 09-August-2012)*.

SCHNEIDER, J., BOLDTE, T., AND WESTERMANN, R. 2006. Real-time editing, synthesis, and rendering of infinite landscapes on gpus. In *Vision, modeling, and visualization 2006: proceedings, November 22-24, 2006, Aachen, Germany*, IOS Press, 145.

SHAO, Y. 2008. *Physics and modelling of wind erosion*, vol. 37. Springer Verlag.

TZENG, S., AND WEI, L. 2008. Parallel white noise generation on a gpu via cryptographic hash. In *Proceedings of the 2008 symposium on Interactive 3D graphics and games*, ACM, 79–87.